Notas de aula 1

Introdução à análise de algoritmos

Prof. Élder F. F. Bernardi (elder.bernardi@passofundo.ifsul.edu.br)
Estrutura de dados III - Ciência da Computação
IFSul - Câmpus Passo Fundo

1. Algoritmos além da solução básica de um problema

Até então a solução algorítmica de um problema bastava-se quando simplesmente solucionava o problema, dado um conjunto mínimo de testes. No entanto, para problemas que demandam um escala maior e/ou uma reutilização constante, esta abordagem não é suficiente, ou melhor: não é **eficiente.** Assim sendo, é preciso ser capaz de analisar como os algoritmos se comportam diante do problema, no que diz respeito a sua solução, compreensão e eficiência. Para um algoritmo ser considerado efetivo, deve seguir as seguintes requisitos:

- ser correto para todos os casos;
- ser compreensível;
- ser eficiente.

Este último requisito é de difícil provação. O que é ser eficiente, quais são os parâmetros que devem ser utilizados e como aferir esta eficiência? Por isso, dentro da Ciência da Computação esta tarefa é papel de uma área completamente dedicada a analisar e avaliar algoritmos: a Análise e Complexidade de Algoritmos. Esta disciplina dedica-se a produzir instrumentos e conhecimentos a fim de avaliar algoritmos e, logicamente, apoiar a construção destes. Neste escopo, necessita-se de um conhecimento mínimo de análise, principalmente no que diz respeito à complexidade do algoritmo. Segue um resumo para a compreensão necessária.

2. O que é complexidade de Algoritmos

A complexidade de um algoritmo, a grosso modo, pode ser traduzida em como este algoritmo se comporta de acordo com sua entrada de dados e seu comportamento no que diz respeito ao consumo de recursos.

Os principais tipos de complexidade são:

- tempo
- espaço
- recursos (rede, cpu, memória...)

Analisar essa complexidade é de suma importância para aferir a eficiência de um algoritmo, para se prever o seu comportamento, poder compará-lo com parâmetros e

metodologias adequados com outras soluções; e, principalmente como uma ferramenta científica para prova (formal) da eficiência do algoritmo.

2.1 Analisando um algoritmo: tipos de entrada e funções de crescimento

O primeiro passo para a análise do algoritmo é observar a **entrada** deste. Entende-se como entrada o conjunto de dados iniciais sobre o qual o algoritmo operará. Neste sentido, a primeira análise diz respeito à "dificuldade" da entrada. Por dificuldade, entende-se o tempo ou uso de recursos que o algoritmo consumirá para resolver o problema. Para fins gerais, o foco será a análise do tempo.

Como exemplo, vamos supor um algoritmo de ordenação básico. Se tivermos uma entrada 1 2 3 4 5, é evidente que o trabalho do algoritmo será menor do que uma entrada 2 3 5 1 4 e menor ainda do que uma entrada 5 4 3 2 1[1].

Os casos de entrada de algoritmos são conhecidos como:

- **Big-O (O)**- O pior caso possível;
- **Theta** (Θ) - Caso médio;
- **Ômega (Ω)** - melhor caso;

Como para se medir a eficiência de uma solução não se pode prever uma entrada média, para fins de análise de algoritmos, **sempre se leva em conta o pior caso**. Desta forma, utiliza-se o que se chama de notação assintótica Big-O, que diz qual é a função de crescimento de um algoritmo com base no seu pior caso de entrada.

Tal notação é expressa da seguinte forma: O(função de crescimento com base na entrada). Isto significa como o tempo de execução de um algoritmo cresce em função do tamanho da sua entrada. Nestes casos, geralmente *n* representa o tamanho da entrada.

Segue a relação das principais funções de crescimento em ordem de complexidade, com exemplos de classes de algoritmos em cada complexidade:

O(1)

Operações simples como *var a=10; a = a*a;* também podem ser computados acessos a memória e operações de controle de fluxo (if, else…);

O(log n)

Problemas que dividem o espaço em dois a cada iteração. Exemplo: busca em uma árvore binária. 16 nodos, leva log(16) passos (4) para encontrar qualquer elemento na árvore.

O(n)

Laços de repetição em geral. Por exemplo: *for (i =0 até n)*

---

[1] Esta análise intuitiva é bastante primitiva e serve somente para uma compreensão inicial. Para fins formais uma análise mais complexa é necessária.

O(n log n)

Operações de laço onde cada elemento é percorrido, porém a cada iteração há uma série decrescente de outras operações a cada iteração. Um exemplo desta classe de algoritmos é o Quick Sort

O(n²)

Operações que envolvem laços aninhados, como por exemplo

```
for (i =0 até n)
    for(j=0 até n)
        fazAlgo();
```

O algoritmo Bubble Sort é um representante desta classe.

$O(2^n)$

Geralmente envolvem soluções recursivas que resolvem um problema de tamanho N, dividindo recursivamente em problemas de N reduzido. Exemplos: solução da Torre de Hanoi. Sequência de fibonacci com um algoritmo recursivo “burro”.

O(n!)

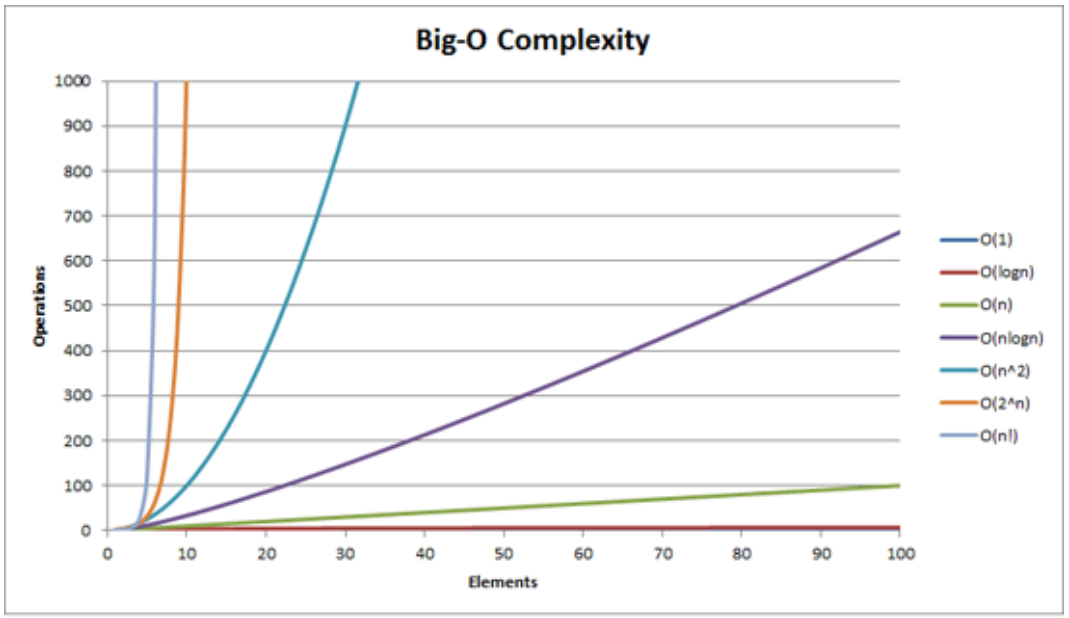
Problemas de arranjo, escalonamento de itens. Há N formas de arranjar um conjunto de itens a cada iteração. Ex.: n=10; na 1a iteração há n formas, depois n-1 e assim sucessivamente. Como para cada decisão tomada em um passo vai interferir em todos os outros, temos a multiplicação,por isso o fatorial: 10*9*8...*1. O Problema do Binpacking é bom exemplo desta classe de algoritmos.

A Figura 1 mostra a plotagem da função de cada classe em relação a *n*. A planilha disponível no rodapé[2] permite testar e visualizar alguns graus de crescimento em cada classe.

---

2 https://docs.google.com/spreadsheets/d/1ZEiDojFOGj01JwE-nR72_ZB2bMMV2LUoDnuRpEHsEoU/edit?usp=sharing

Figura 1: crescimento assintótico das principais classes de algoritmos



Fonte: https://i.stack.imgur.com/Aq09a.png

2.2 Notas importantes sobre classificação de algoritmos

Para se prosseguir na análise dos algoritmos, tem-se que ter em mente dois conceitos bastante importantes: determinação da complexidade assintótica do algoritmo e capacidade de solução deste à medida que o seu n (entrada) tende ao infinito. As duas coisas demandam um pouco mais de estudo e aqui são apresentadas somente de forma introdutória.

2.2.1 Determinação da complexidade (Big-O)

Se pegarmos por exemplo um algoritmo bem simples com as seguintes instruções, onde comentado está o custo de cada operação:

```
a=2;  //1
for (i =0 até n){ // n
        a *=2; //1
        a *=3; //1
        for(j=0 até n) //n
                a+=j; //1
}
```

Se formos “somar” o total de operações do algoritmo, teríamos algo como: *1 + n * 1+1 * n *1*. Note que a variável *n* indica repetição de operações sobre o tamanho da entrada, enquanto que operações simples são constantes. Se fossemos “somar” todas as operações poderíamos dizer que temos: O(1+ n * 2 * n *1), simplicando as multiplicações: O(1 +n^2* 2). Uma regra bastante importante aqui é a seguinte: **se *n* for muito grande, tendendo ao infinito, todas as constantes multiplicadoras ou somadoras são**

***irrelevantes*. Isto na prática significa que a complexidade de um algoritmo deve ser levada sempre em conta pela sua parte mais complexa, no pior caso possível.** Então no exemplo pode-se afirmar que a complexidade do algoritmo é O(n^2).

Mesmo se acrescentássemos outro *loop* aninhado (o que faria a complexidade ser O(n^3)), pode-se continuar dizendo que é O(n^2), pois com um n → ∞ tanto faz se o seu expoente é 2, 3 ou 1000. Note que se for 1, faz diferença, pois aí a complexidade seria O(n).

De modo bem prático, ao analisar a complexidade de um algoritmo analise todos os códigos, os n e constantes e faça a multiplicação e/ou soma dos tempos. O pior tempo possível será o O().

2.2.2 Problemas P e NP-Completos

Há uma classificação de problemas na computação que serve para determinar o grau de solubilidade de um problema. Simplificando, o quanto um problema é solucionável em um tempo possível, à medida que seu *n* cresça. Com isto, afirma-se que um problema é da classe N quando há uma solução algorítmica para este em um tempo computacional *polinomial*. Os problemas NP-Completos são aqueles que não possuem uma solução possível em um tempo computacional *não polinomial.* Há toda uma teoria (bem complicada) que demonstra tudo isto, beirando até a filosofia, mas não é o escopo do momento. O que é importante aprender agora é que todos os problemas que somente tem soluções possíveis conhecidas cujas complexidades são exponenciais, são considerados problemas **sem solução computacional possível**. Isto significa que não há como solucioná-los de forma determinística como um teorema, é preciso uma heurística para tal. Por exemplo, se alguém lhe perguntar para desenvolver um algoritmo de roteamento de rede que encontre SEMPRE o melhor caminho para um pacote na Internet, você deve saber que não há como solucioná-lo, pois a sua solução determinística consumiria um tempo maior para ser encontrada do que se você roteasse o pacote de forma aleatória. Então só lhe resta buscar uma solução mais simples e que não dá nenhuma certeza se o resultado encontrado é de fato o melhor possível, mas resolve o problema em um tempo aceitável, com uma solução satisfatória dentro das limitações: uma heurística.

Para fins de informação, todos os problemas que somente tem soluções conhecidas dentro com complexidade O(2^n) ou piores, são considerados NP-Completos, ou seja, somente terão uma solução determinística computacionalmente viável se suas entradas sejam razoavelmente pequenas.

Para aprofundar-se, busque o problema do *caixeiro viajante* e todas as suas implicações…

Conclusão (ou o que tudo isso quer dizer, enfim)

Tudo isso serve para que basicamente sejamos capazes de identificar soluções ruins para problemas solúveis para não perdermos tempo com elas e gastarmos energia na busca de soluções mais otimizadas; e que também possamos identificar problemas insolúveis para não perdermos tempo tentando encontrar a solução perfeita para eles e ao invés disto, gastarmos tempo procurando heurísticas viáveis. Isto se faz através da análise

de algoritmos. Com ela é possível ver se a solução é computacionalmente viável e, de acordo com a classe do problema, buscar **otimizações** que tornem a solução adequada.

É com esta ferramenta em mãos que se prossegue no foco da disciplina: o estudo de algoritmos de busca e seleção que sejam viáveis e eficientes e em que situações devemos optar por um ou outro.

ESTE MATERIAL ESTÁ DISPONÍVEL EM: https://goo.gl/KHKqkr